Using Datalog and Boolean Equation Systems for Program Analysis *

María Alpuente, Marco A. Feliú, Christophe Joubert, and Alicia Villanueva

Universidad Politécnica de Valencia, DSIC / ELP
Camino de Vera s/n, 46022, Valencia, Spain
{alpuente,mfeliu,joubert,villanue}@dsic.upv.es

Abstract. This paper describes a powerful, fully automated method to evaluate Datalog queries by using Boolean Equation Systems (BESs), and its application to object-oriented program analysis. Datalog is used as a specification language for expressing complex interprocedural program analyses involving dynamically created objects. In our methodology, Datalog rules encoding a particular analysis together with a set of constraints (Datalog facts that are automatically extracted from program source code) are dynamically transformed into a BES, whose local resolution corresponds to the demand-driven evaluation of the program analysis. This approach allows us to reuse existing general purpose verification toolboxes, such as CADP, providing local BES resolutions with linear-time complexity. Our evaluation technique has been implemented and successfully tested on several JAVA programs and Datalog analyses that demonstrate the feasibility of our approach.

Keywords: program analysis, Datalog, boolean equation system, demand-driven evaluation

1 Introduction

Program analysis is a technique for statically determining dynamic properties of programs. Static analysis generally executes an abstract version of the program's semantics on abstract data, rather than on concrete data. While originally established as a technique used in optimizing compilers, program analysis is also commonly used in software-development tools that help to find program errors and also derive safety properties of programs.

Recently, a large number of program analyses have been developed in Datalog [15,18], a simple relational query language rich enough to describe complex interprocedural program analyses involving dynamically created objects.

The advantages of formulating dataflow analyses as a Datalog query are twofold. On the one hand, analyses that take hundreds of lines of code in a traditional language can be expressed in a few lines of Datalog [18]. On the other hand, an important number of optimization techniques for Datalog have been studied extensively in logic programming and deductive databases [1,4]. The two general approaches for evaluating

* This work has been supported by the Spanish MEC under grant TIN2007-68093-C02-02, by the Generalitat Valenciana GVPRE/2008/113, and by the Universidad Politécnica de Valencia, under grant PAID-06-07 (TACPAS).

Datalog queries are the top-down and the bottom-up methods. Given a set of rules, the bottom-up approach computes all facts that can be inferred from the program and then selects those that unify with the given query. The top-down, goal-directed approach computes on-demand. While bottom-up computation may be very inefficient, the top-down approach is prone to infinite loops and redundant computations. Optimization methods for both approaches that resolve the major drawbacks have been developed, such as bottom-up transformations based on magic sets [3] and top-down evaluation with tabling [4]. In the *Query-Sub-Query* (QSQ) optimization technique [16], goals are generated top-down, but whenever possible, goals are propagated in sets at a time, rather than one at a time, and all generated goals and facts are memoized.

This paper describes the use of Boolean Equation Systems (BES) [2] to evaluate Datalog queries and its application to object-oriented program analysis. Our technique is based on top-down evaluation guided by the given query, and makes use of tables and finite data domains to ensure termination. Our method is not a direct evaluation method because it transforms the rules prior to evaluate them. Similarly to the QSQ technique [16], computation is done by proceeding with a set tuples at a time. This can be a great advantage for large datasets since it makes disk accesses more efficient. In our program analysis methodology, Datalog rules encoding a particular analysis, together with a set of constraints (Datalog facts that are automatically extracted from program source code), are dynamically transformed into a BES, whose local resolution corresponds to the demand-driven evaluation of the program analysis. This approach allows us to reuse existing general purpose verification toolboxes, such as CADP, providing local BES resolutions with linear-time complexity.

Related Work. The description of data-flow analyses as a database query was pioneered by Ullman [15] and Reps [13] who applied Datalog's bottom-up magic-set implementation to automatically derive a *local* implementation.

Recently, BESs with typed parameters [11], called PBES, have been successfully used to encode several hard verification problems such as the first-order value-based modal μ-calculus model-checking problem [12], and the equivalence checking of various bisimulations [5] on (possibly infinite) labeled transition systems. However, PBESs have not yet been used to compute complex interprocedural program analyses involving dynamically created objects.

The closest related work proposes the use of Dependency Graphs (DGs) for representing satisfaction problems, including propositional Horn Clauses satisfaction and BES resolution [10]. A linear time algorithm for propositional Horn Clauses satisfiability is described in terms of the least solution of a DG equation system. This corresponds to an alternation-free BES, which can only deal with propositional logic problems. The extension of Liu and Smolka's work [10] to Datalog query evaluation is not straightforward. This is testified by the encoding of data-based temporal logics in equation systems with parameters in [12], where each boolean variable may depend on multiple data terms. DGs are not sufficiently expressive to represent such data dependencies on each vertex. Hence, it is necessary to work at a higher level, on the PBES representation.

Recently, a very efficient Datalog program analysis technique based on binary decision diagrams (BDDs) has been developed in the BDDBDDB system [18], which scales

to large programs and is competitive w.r.t. the traditional (imperative) approach. The computation is achieved by a fixed point computation starting from the everywhere false predicate (or some initial approximation based on Datalog facts). Datalog rules are then applied in a bottom-up manner until saturation is reached, so that all solutions satisfying each relation of a Datalog program are exhaustively computed. These sets of solutions are then used to answer complex formulas.

In contrast, our approach focus on demand-driven techniques to solve a set of queries with no *a priori* computation of the derivable atoms. In the context of program analysis, note that all program updates, like pointer updates, might potentially be inter-related, leading to an exhaustive computation of all results. Therefore, improvements to top-down evaluation remain attractive for program analysis applications. Recently, Zheng and Rugina [19] showed that demand-driven CFL-reachability with worklist algorithm can compare favorably with an exhaustive solution, especially in terms of memory consumption. Our technique to solve Datalog programs based on local BES resolution goes towards the same direction and provides a novel approach to demand-driven program analyses.

Plan of the Paper. The rest of the paper is organized as follows: Section 2 recalls Datalog definitions and the BES formalism with its parameterised extension. Our methodology to transform Datalog query to an implicit BES with parameters is described in Section 3. Section 4 illustrates the application of Datalog and BES to program analysis, together with experimental results on JAVA programs and context-insensitive pointer analysis. Finally, Section 5 concludes and highlights future research directions.

2 Preliminaries

2.1 Datalog

Datalog [15] is a relational language using declarative *rules* to both describe and query a deductive database. A Datalog rule is a function-free Horn clause over an alphabet of *predicate* symbols (e.g. relation names or arithmetic predicates, such as $<$) whose *arguments* are either variables or constant symbols. A *Datalog program* $\mathcal{R}$ is a finite set of Datalog rules.

Definition 1 (Syntax of Rules). *Let $\mathcal{P}$ be a set of* predicate *symbols, $\mathcal{V}$ be a finite set of* variable *symbols, and $\mathcal{C}$ a set of* constant *symbols. A Datalog* rule *r, also called* clause*, defined over a finite alphabet $P \subseteq \mathcal{P}$ and arguments from $V \cup C$, $V \subseteq \mathcal{V}$, $C \subseteq \mathcal{C}$, has the following syntax:*

$$p_0(a_{0,1}, \ldots, a_{0,n_0}) \ :- \ p_1(a_{1,1}, \ldots, a_{1,n_1}), \ \ldots, \ p_m(a_{m,1}, \ldots, a_{m,n_m}).$$

where each p_i is a predicate symbol of arity n_i with arguments $a_{i,j} \in V \cup C$ ($j \in [1..n_i]$).

The atom $p_0(a_{0,1}, \ldots, a_{0,n_0})$ in the left-hand side of the clause is the rule's *head*, where p_0 is neither arithmetic nor negated. The finite conjunction of *subgoals* in the right-hand side of the formula is the rule's *body*, *i.e.*, atoms that may optionally be negated or arithmetic, and contain all variables appearing in the head. Following logic

programming terminology, a rule with empty body ($m = 0$) is called a *fact* whereas a rule with empty head and $m > 0$ is called a *goal*. To keep the presentation simple, we restrict our syntax to predicate symbols of arity 1. A syntactic object (argument, atom, or rule) that contains no variables is called *ground*. The *Herbrand Universe* of a Datalog program R defined over P, V and C, denoted U_R, is the finite set of all ground arguments, *i.e.*, constants of C. The *Herbrand Base* of R, denoted B_R, is the finite set of all ground atoms that can be built by assigning elements of U_R to the predicate symbols in P. A *Herbrand Interpretation* of R, denoted I (from a set $\mathcal{I}$ of Herbrand interpretations, $\mathcal{I} \subseteq B_R$), is a set of ground atoms.

Definition 2 (Fixed point semantics). *Let R be a Datalog program. The least* Herbrand model *of R is a Herbrand interpretation I of R defined as the least fixed point of a monotonic, continuous operator $T_R : \mathcal{I} \to \mathcal{I}$ known as the* immediate consequences operator *and defined by:*

$$T_R(I) = \{h \in B_R \mid \ h : -b_1, ..., b_m \text{ is a ground instance of a rule in } R, \\ \text{with } \ b_i \in I, i = 1..m, m \geq 0\}$$

Note that T_R computes both, ground atoms derived from applicable rules–called *intentional database* (or *idb*)–, and ground instances of rules with an empty body ($m = 0$), also called *extensional database* (*edb*). The choice of minimal model as the semantics of a Datalog program is justified by the assumption that all facts that are not in the database are false.

The number of Herbrand models being finite for a Datalog program R, there always exists a least fixed point for T_R, denoted μT_R, which is the least Herbrand model of R. In practice, one is generally interested in the computation of some specific atoms, called *queries*, and not in the whole database of atoms. Hence, queries may be used to prevent the computation of facts that are irrelevant for the atoms of interest, *i.e.*, facts that are not derived from the query.

Definition 3 (Query Evaluation). *A* Datalog query *q is a pair $\langle G, R\rangle$ where:*

- *R is a Datalog program defined over P, V and C,*
- *G is a set of* goals.

Given a query q, its evaluation *consists in computing $\mu T_{\{q\}}$, $\{q\}$ being the extension of the Datalog program R with the Datalog rules in G.*

The evaluation of a Datalog program augmented with a set of goals deduces all the different constant combinations that, when assigned to the variables in the goals, can make one of the goal clauses true, *i.e.*, all atoms b_i in its body are satisfied.

2.2 Parameterised Boolean Equation System

Given $\mathcal{X}$ a set of boolean variables and $\mathcal{D}$ a set of data terms, a *Parameterised Boolean Equation System* [11] (PBES) $B = (x_0, M_1, ..., M_n)$ is a set of n blocks M_i, each one containing $p_i \in \mathbb{N}$ fixed-point equations of the form

$$x_{i,j}(\vec{d}_{i,j} : \vec{D}_{i,j}) \stackrel{\sigma_i}{=} \phi_{i,j}$$

with $j \in [1..p_i]$ and $\sigma_i \in \{\mu, \nu\}$, also called *sign* of equation i, the least (μ) or greatest (ν) fixed point operator. Each $x_{i,j}$ is a boolean variable from $\mathcal{X}$ that binds zero or more data terms $d_{i,j}$ of type $D_{i,j}$[1] which may occur in the *boolean formula* $\phi_{i,j}$ (from a set Φ of boolean formulae). $x_0 \in \mathcal{X}$, defined in block M_1, is a boolean variable whose value is of interest in the context of the local resolution methodology. Boolean formulae $\phi_{i,j}$ are formally defined as follows.

Definition 4 (Boolean Formula). *A* boolean formula ϕ*, defined over an alphabet of (parameterised) boolean variables* $X \subseteq \mathcal{X}$ *and data terms* $D \subseteq \mathcal{D}$*, has the following syntax given in positive form:*

$$\phi, \phi_1, \phi_2 ::= \mathsf{true} \mid \mathsf{false} \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid X(e) \mid \forall d \in D.\ \phi \mid \exists d \in D.\ \phi$$

where boolean constants and operators have their usual definition, e *is a data term (constant or variable of type* D*),* $X(e)$ *denotes the* call *of a boolean variable* X *with parameter* e*, and* d *is a term of type* D*.*

A *boolean environment* $\delta \in \Delta$ is a partial function mapping each (parameterised) boolean variable $x(d : D)$ to a predicate $\delta(x) : \mathcal{X} \to (D \to \mathbb{B})$, with $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$. *Boolean constants* true and false abbreviate the empty conjunction $\wedge\emptyset$ and the empty disjunction $\vee\emptyset$ respectively. A *data environment* $\varepsilon \in \mathcal{E}$ is a partial function mapping each data term e of type D to a value $\varepsilon(e) : D \to D$, which forms the so-called support of ε, noted $supp(\varepsilon)$. Note that $\varepsilon(e) = e$ when e is a constant data term. The *overriding* of ε_1 by ε_2 is defined as $(\varepsilon_1 \oslash \varepsilon_2)(x) = \mathit{if}\ x\ \in supp(\varepsilon_2)\ \mathit{then}\ \varepsilon_2(x)\ \mathit{else}\ \varepsilon_1(x)$. The *interpretation function* $[\![\phi]\!]\delta\varepsilon$, where $[\![.]\!] : \Phi \to \Delta \to \mathcal{E} \to \mathbb{B}$, gives the truth value of boolean formula ϕ in the context of δ and ε, where all free boolean variables x are evaluated by $\delta(x)$, and all free data terms d are evaluated by $\mathcal{E}(d)$.

Definition 5 (Semantics of Boolean Formula). *Let* $\delta : \mathcal{X} \to (D \to \mathbb{B})$ *be a* boolean environment *and* $\varepsilon : D \to D$ *be a data environment. The semantics of a boolean formula* ϕ *is inductively defined by the following interpretation function:*

$$\begin{aligned}
[\![\mathsf{true}]\!]\delta\varepsilon &= \mathsf{true} \\
[\![\mathsf{false}]\!]\delta\varepsilon &= \mathsf{false} \\
[\![\phi_1 \wedge \phi_2]\!]\delta\varepsilon &= [\![\phi_1]\!]\delta\varepsilon \wedge [\![\phi_2]\!]\delta\varepsilon \\
[\![\phi_1 \vee \phi_2]\!]\delta\varepsilon &= [\![\phi_1]\!]\delta\varepsilon \vee [\![\phi_2]\!]\delta\varepsilon \\
[\![x(e)]\!]\delta\varepsilon &= (\delta(x))(\varepsilon(e)) \\
[\![\forall d \in D.\ \phi]\!]\delta\varepsilon &= \forall\, v \in D,\ [\![\phi]\!]\delta(\varepsilon \oslash [v/d]) \\
[\![\exists d \in D.\ \phi]\!]\delta\varepsilon &= \exists\, v \in D,\ [\![\phi]\!]\delta(\varepsilon \oslash [v/d])
\end{aligned}$$

Definition 6 (Semantics of Equation Block). *Given a* PBES $B = (x_0, M_1, ..., M_n)$ *and a boolean environment* δ*, the solution* $[\![M_i]\!]\delta$ *to a block* $M_i = \{x_{i,j}(d_{i,j} : D_{i,j}) \overset{\sigma_i}{=} \phi_{i,j}\}_{j \in [1,p_i]}$ *(*$i \in [1..n]$*) is defined as follows:*

$$[\![\{x_{i,j}(d_{i,j} : D_{i,j}) \overset{\sigma_i}{=} \phi_{i,j}\}_{j \in [1,p_i]}]\!]\delta = \sigma_i \Psi_{i\delta}$$

[1] To simplify our description in the rest of the paper, we intentionally restrict to one the maximum number of data term parameter $d : D$.

where $\Psi_{i\delta} : (D_{i,1} \to \mathbb{B}) \times \ \ldots \ \times \ (D_{i,p_i} \to \mathbb{B}) \to (D_{i,1} \to \mathbb{B}) \times \ \ldots \ \times \ (D_{i,p_i} \to \mathbb{B})$ *is a vectorial functional defined as*

$$\Psi_{i\delta}(g_1, ..., g_{p_i}) = (\lambda v_{i,j} : D_{i,j}.[\![\phi_{i,j}]\!](\delta \oslash [g_1/x_{i,1}, ..., g_{p_i}/x_{i,p_i}])[v_{i,j}/d_{i,j}])_{j\in[1,p_i]}$$

where $g_i : D_i \to \mathbb{B}$, $i \in [1..p_i]$.

A Pbes is *alternation-free* if there are no mutual recursion between boolean variables defined by least ($\sigma_i = \mu$) and greatest ($\sigma_i = \nu$) fixed point boolean equations. In this case, equation blocks can be sorted topologically such that the resolution of a block M_i only depends upon variables defined in a block M_k with $i < k$. A block M_i is *closed* when the resolution of all its boolean formulae $\phi_{i,j}$ only depends upon boolean variables $x_{i,k}$ from M_i.

Definition 7 (Semantics of alternation-free PBES). *Given an alternation-free* Pbes $B = (x_0, M_1, ..., M_n)$ *and a boolean environment* δ*, the semantics* $[\![B]\!]\delta$ *to* B *is the value of its main variable* x_0 *given by the semantics of* M_1, i.e., $\delta_1(x_0)$*, where the contexts* δ_i *are calculated as follows:*

$$\begin{array}{l} \delta_n = [\![M_n]\!][] \text{ (the context is empty because } M_n \text{ is closed)} \\ \delta_i = ([\![M_i]\!]\delta_{i+1}) \oslash \delta_{i+1} \text{ for } i \in [1, n-1] \end{array}$$

where each block M_i *is interpreted in the context of all blocks* M_k *with* $i < k$.

3 Datalog Queries and Boolean Equation Systems

An elegant and direct intermediate representation of a Datalog query can be given as an implicit Bes parameterised with typed boolean variables. In this section, we present reductions between Datalog query evaluation and Pbes resolution for both directions of reducibility. The reductions are linear-time with a suitable representation of the problem instances. As in [18], we assume that Datalog programs have stratified negation (no recursion through negation), and totally-ordered finite domains, without considering comparison operators.

3.1 Datalog query representation

We propose a transformation of the Datalog query into a related query, expressed as a parameterised boolean variable of interest and a Pbes, which is subsequently evaluated using traditional Pbes evaluation techniques.

Proposition 1. *Let* $q = \langle G, R\rangle$ *be a Datalog query, defined over* P*,* V *and* C*, and* $B_q = (x_0, M_1)$*, with* $\sigma_1 = \mu$*, a* Pbes *defined over a set* $\mathcal{X}$ *of boolean variables* x_p *in one-to-one correspondence with predicate symbols* p *of* P *plus a special variable* x_0*, a set* $\mathcal{D}$ *of data terms in one-to-one correspondence with variable and constant symbols of* $V \cup C$*, and* M_1 *the block containing exactly the following equations, where fresh*

variables are existentially quantified after the transformation:

$$x_0 \stackrel{\mu}{=} \bigvee_{:-\ q_1(d_1),\ \ldots,\ q_m(d_m).\ \in G} \bigwedge_{i:=1}^{m} x_{q_i}(d_i) \tag{1}$$

$$\{x_p(d:D) \stackrel{\mu}{=} \bigvee_{p(d)\ :-\ p_1(d_1),\ldots\ p_m(d_m).\ \in R} \bigwedge_{i:=1}^{m} x_{p_i}(d_i) \mid p \in P\} \tag{2}$$

Then q is satisfiable if and only if $[\![B]\!]\delta(x_0) = \mathsf{true}$.

Boolean variable x_0 encodes the set of Datalog goals G, whereas (paremeterised) boolean variables $x_p(d:D)$ represent the set of Datalog rules R modulo renaming.

In our framework, the reverse direction of reducibility consists in the transformation of a parameterised boolean variable of interest, defined in a PBES, into a related relation of interest expressed as a Datalog query, which could be evaluated using traditional Datalog evaluation techniques.

Proposition 2. *Let $B = (x_0, M_1)$, with $\sigma_1 = \mu$, be a* PBES *defined over a set $\mathcal{X}$ of boolean variables and a set $\mathcal{D}$ of data terms, and $q_B = \langle G, R\rangle$ be a Datalog query defined over a set P of predicate symbols p in one-to-one correspondence with boolean variables x_p of $\mathcal{X} \setminus \{x_0\}$, a set $V \cup C$ of variable and constant symbols in one-to-one correspondence with data terms of D, and $\langle G, R\rangle$ containing exactly the following Datalog rules:*

$$G = \left\{ \begin{array}{l} :- \quad q_{1,1}(d_{1,1}), \ldots, \quad q_{1,n_j}(d_{1,n_j})., \\ \qquad\qquad \vdots \\ :- \ q_{n_i,1}(d_{n_i,1}), \ldots, q_{n_i,n_j}(d_{n_i,n_j}). \end{array} \middle| \ x_0 \stackrel{\mu}{=} \bigvee_{i=1}^{n_i} \bigwedge_{j=1}^{n_j} x_{q_{i,j}}(d_{i,j}) \in M_1 \right\}$$

$$R = \left\{ \begin{array}{l} p(d) :- \quad p_{1,1}(d_{1,1}), \ldots, \quad p_{1,n_j}(d_{1,n_j})., \\ \qquad\qquad \vdots \\ p(d) :- \ p_{n_i,1}(d_{n_i,1}), \ldots, \ p_{n_i,n_j}(d_{n_i,n_j}). \end{array} \middle| \ x_p(d) \stackrel{\mu}{=} \bigvee_{i=1}^{n_i} \bigwedge_{j=1}^{n_j} x_{p_{i,j}}(d_{i,j}) \in M_1 \right\}$$

Then $[\![B]\!]\delta(x_0) = \mathsf{true}$ if and only if $q_B = \langle G, R\rangle$ is satisfiable.

Example 1. We illustrate the reduction method from Datalog to PBES by means of a simple Datalog example. Let $q = \langle G, R\rangle$ be the following Datalog query with $D = \{mary, alice, mark, X, Y, Z\}$:

```
:- superior (mary, Y).
supervise(mary, alice).
supervise(alice, mark).
superior(X, Y) :- supervise(X, Y).
superior(X, Y) :- supervise(X, Z), superior(Z, Y).
```

By using Proposition 1, we obtain the following PBES:

$$\begin{array}{ll} x_0 & \stackrel{\mu}{=} \exists Y \in D\,.\, x_{superior}(mary, Y) \\ x_{supervise}(mary, alice) & \stackrel{\mu}{=} \mathsf{true} \\ x_{supervise}(alice, mark) & \stackrel{\mu}{=} \mathsf{true} \\ x_{superior}(X:D, Y:D) & \stackrel{\mu}{=} x_{supervise}(X, Y) \vee \\ & \qquad \exists Z \in D.(x_{supervise}(X, Z) \wedge x_{superior}(Z, Y)) \end{array}$$

In the rest of this paper, we will develop the use of PBESs to solve Datalog queries.

3.2 Instantiation to parameterless BES

Among the different known techniques for solving a PBES, such as Gauss elimination with symbolic approximation, and use of patterns, under/over approximations, or invariants, we consider the resolution method based on transforming the PBES into a parameterless boolean equation system (BES) that can be solved by linear time and memory algorithms [11,7] when data domains are finite.

Definition 8 (Boolean Equation System). *A* Boolean Equation System *(*BES*)* $B = (x_0, M_1, ..., M_n)$ *is a* PBES *where data domains are removed and boolean variables, being independent from data parameters, are considered* propositional.

To obtain a direct transformation into a parameterless BES, we first described the PBES in a simpler format. This simplification step consists in introducing new variables, such that each formula at the right-hand side of a boolean equation only contains at most one operator. Hence, boolean formulae are restricted to pure disjunctive or conjunctive formulae.

Given a Datalog query $q = \langle G, R \rangle$, by applying this simplification to the PBES of Proposition 1, we obtain the following PBES:

$$x_0 \stackrel{\mu}{=} \bigvee_{:-\ q_1(d_1),...,q_m(d_m).\ \in G} g_{q_1(d_1),...,q_m(d_m)}$$

$$g_{q_1(d_1),...,q_m(d_m)} \stackrel{\mu}{=} \bigwedge_{i:=1}^{m} x_{q_i}(d_i)$$

$$x_p(d : D) \stackrel{\mu}{=} \bigvee_{p(d)\ :-\ p_1(d_1),...,p_m(d_m).\ \in R} r_{p_1(d_1),...,p_m(d_m)}$$

$$r_{p_1(d_1),...,p_m(d_m)} \stackrel{\mu}{=} \bigwedge_{i:=1}^{m} x_{p_i}(d_i)$$

By applying the instantiation algorithm of Mateescu [11], we eventually obtain a parameterless BES, where all possible values of each typed data terms have been enumerated over their corresponding finite data domains.

The resulting implicit parameterless BES is defined as follows, where $\preceq$ is the standard preorder of relative generality (instantiation ordering).

$$x_0 \stackrel{\mu}{=} \bigvee_{:-\ q_1(d_1),...,q_m(d_m).\ \in G} g_{q_1(d_1),...,q_m(d_m)} \tag{3}$$

$$g_{q_1(d_1),...,q_m(d_m)} \stackrel{\mu}{=} \bigvee_{1 \leq i \leq m,\ e_i \in D_i \wedge d_i \preceq e_i} g^c_{q_1(e_1),...,q_m(e_m)} \tag{4}$$

$$g^c_{q_1(e_1),...,q_m(e_m)} \stackrel{\mu}{=} \bigwedge_{i:=1}^{m} x_{q_{i_{e_i}}} \tag{5}$$

$$x_{p_d} \overset{\mu}{=} \bigvee_{p(d)\ :-\ p_1(d_1),\ldots,p_m(d_m).\ \in R} r_{p_1(d_1),\ldots,p_m(d_m)} \tag{6}$$

$$r_{p_1(d_1),\ldots,p_m(d_m)} \overset{\mu}{=} \bigvee_{1\leq i\leq m,\, e_i\in D_i \wedge d_i \preceq e_i} r^c_{p_1(e_1),\ldots,p_m(e_m)} \tag{7}$$

$$r^c_{p_1(e_1),\ldots,p_m(e_m)} \overset{\mu}{=} \bigwedge_{i:=1}^{m} x_{p_{i_{e_i}}} \tag{8}$$

Observe that Equation 1 is transformed into a set of parameterless equations (3, 4, 5). First, Equation 3 describes the set of parameterised goals $g_{q_1(d_1),\ldots,q_m(d_m)}$ of the query. Then, Equation 4 represents the instantiation of each variable parameter d_i to the possible values e_j from the domain. Finally, Equation 5 states that each instantiated goal $g^c_{q_1(e_1),\ldots,q_m(e_m)}$ is satisfied whenever the values e_j make all predicates q_i of the goal `true`. Similarly, Equation 2 (describing Datalog rules) is encoded into a set of parameterless equations (6, 7, 8).

3.3 Optimizations

The parameterless Bes described above is inefficient since it adopts a brute-force approach that, in the very first steps of the computation (Equation 4), enumerates all possible tuples of the query. It is well-known that a Datalog program runs in $O(n^k)$ time, where k is the largest number of variables in any single rule, and n is the number of constants in the facts and rules. Similarly, for a simple query like `:- superior(X,Y).`, with `X` and `Y` elements of a domain D of size 10 000, Equation 4 will generate D^2, *i.e.*, 10^8, boolean variables representing all possible combinations of values `X` and `Y` in relation `superior`. Usually, for each atom in a Datalog program, the number of facts that are given or inferred by the Datalog rules is much lower than the *domain's size* to the power of *atom's arity*. Ideally, the Datalog query evaluation should enumerate (given or inferred) facts only *on-demand*.

Among the existing optimizations for top-down evaluation of Datalog queries, the so-called *Query-Sub-Query* [16] technique consists in minimizing the number of tuples derived by a rewriting of the program based on the propagation of bindings. Basically, the method aims at keeping the bindings of variables between atoms $p(a)$ in a rule. In our Datalog evaluation technique based on Bes, we adopt a similar approach: two boolean equations (Equations 4 and 7 slightly modified) only enumerate the values of variable arguments that appear more than once in the body of the corresponding Datalog rule, otherwise arguments are kept unchanged. Moreover, if the atom $p(a)$ is part of the Extensional Database, the only possible values of its variable arguments are values that reproduce a given fact of the Datalog program. We note D^p_i the subdomain of D that contains all possible values of the i^{th} variable argument of p if p is in Extensional Database, otherwise $D^p_i = D$. Hence, the resulting Bes resolution is likely to process fewer facts and be more efficient than the brute-force approach.

Following this optimization technique, a parameterless Bes can directly be derived from the previous Bes representation which we define as follows:

$$x_0 \overset{\mu}{=} \bigvee_{:-\ q_1(d_1),\ldots,q_m(d_m).\ \in G} g_{q_1(d_1),\ldots,q_m(d_m)} \tag{9}$$

$$g_{q_1(d_1),\ldots,q_m(d_m)} \stackrel{\mu}{=} \bigvee_{\substack{\{a_1,\ \ldots,\ a_m\}\in(\{V\cup D_1^{q_1}\}\times\ldots\times\{V\cup D_1^{q_m}\})\ \mid \\ \textit{if}\ (\exists\ j\in[1..m],\ j\neq i\mid d_i=d_j\wedge d_i\in V)}} g^{pc}_{q_1(a_1),\ldots,q_m(a_m)}$$
$$\textit{then}\ a_i\in D_1^{q_i}\ \wedge\ (\forall\ j\in[1..m],\ d_i=d_j\mid a_j:=a_i)\ \textit{else}\ a_i:=d_i \tag{10}$$

$$g^{pc}_{q_1(a_1),\ldots,q_m(a_m)} \stackrel{\mu}{=} \bigwedge_{i:=1}^{m} x_{q_{i_{a_i}}} \tag{11}$$

$$x_{q_a} \stackrel{\mu}{=} x^f_{q_a} \vee x^r_{q_a} \tag{12}$$

$$x^f_{q_a} \stackrel{\mu}{=} \bigvee_{(e:=a\ \wedge\ a\in C)\ \vee\ (e\in D_1^q\ \wedge\ a\in V)\ \mid q(e).\in R} x^c_{q_e} \tag{13}$$

$$x^c_{q_e} \stackrel{\mu}{=} \mathsf{true} \tag{14}$$

$$x^r_{p_a} \stackrel{\mu}{=} \bigvee_{p(a)\ :-\ p_1(d_1),\ldots,p_m(d_m).\ \in R} r_{p_1(d_1),\ldots,p_m(d_m)} \tag{15}$$

$$r_{p_1(d_1),\ldots,p_m(d_m)} \stackrel{\mu}{=} \bigvee_{\substack{\{a_1,\ \ldots,\ a_m\}\in(\{V\cup D_1^{p_1}\}\times\ldots\times\{V\cup D_1^{p_m}\})\ \mid \\ \textit{if}\ (\exists\ j\in[1..m],\ j\neq i\mid d_i=d_j\wedge d_i\in V)}} r^{pc}_{p_1(a_1),\ldots,p_m(a_m)}$$
$$\textit{then}\ a_i\in D_1^{p_i}\ \wedge\ (\forall\ j\in[1..m],\ d_i=d_j\mid a_j:=a_i)\ \textit{else}\ a_i:=d_i \tag{16}$$

$$r^{pc}_{p_1(a_1),\ldots,p_m(a_m)} \stackrel{\mu}{=} \bigwedge_{i:=1}^{m} x_{p_{i_{a_i}}} \tag{17}$$

Observe that Equations 9, 11, 15 and 16 correspond respectively to Equations 3, 5, 6 and 8 of previous Bes definition with only a slight renaming of generated boolean variables. The important novelty is that, instead of enumerating all possible values of the domain, as it is done in Equation 4, the corresponding new Equation 10 only enumerates the values of variable arguments that are repeated in the body of a rule, otherwise variable arguments are kept unchanged *i.e.*, $a_i := d_i$. Indeed, the generated boolean variables $g^{pc}_{q_1(a_1),\ldots,q_m(a_m)}$ may still refer to atoms containing variable arguments. Thus, the combinatorial explosion of possible tuples is avoided at this point and delayed to future steps. Equation 12 generates two boolean successors for variable x_{q_a}: $x^f_{q_a}$ when q is a relation that is part of the Extensional Database, and $x^r_{q_a}$ when q is defined by Datalog rules. In Equation 13, each value of a (variable or constant) that leads to a given fact $q(e).$ of the program, generates a new boolean variable $x^c_{q_e}$, that is true by definition of a fact. Equation 15 simply infers Datalog rules whose head is p_a. Note that Equations 10, 13, and 16 enumerate possible values of subdomains $D_1^{p_i}$ instead of full domain D. With the Datalog program described in Example 1, this restriction would consist in using two new subdomains $D_1^{supervise} = \{mary, alice\}$ and $D_2^{supervise} = \{alice, mark\}$ instead of domain $D = \{mary, alice, mark\}$ for the values of each variable argument in relation *supervise*.

3.4 Solutions extraction

Considering the optimized parameterless Bes defined above, the query satisfiability problem is reduced to the local resolution of boolean variable x_0. The value (true

or false) computed for x_0 indicates whether there exists at least one satisfiable goal in G. We can remark that the BES representing the evaluation of a Datalog query is only composed of one equation block containing alternating dependencies between disjunctive and conjunctive variables. Hence, it can be solved by optimized depth-first search (DFS) for such a type of equation block. However, since the DFS strategy can only conclude the existence of a solution to the query by computing a minimal number of boolean variables, it is necessary to use a breadth-first search (BFS) strategy to compute all the different solutions of a Datalog query. Such a strategy will "force" the resolution of all boolean variables that have been put in the BFS queue, even if the satisfiability of the query has been computed in the meantime. Consequently, the solver will compute all possible boolean variables $x^c_{q_e}$, which are potential solutions for the query. Upon termination of the BES resolution (ensured by finite data domains and table-based exploration), query solutions, *i.e.*, combinations of variable values $\{e_1, \ldots, e_m\}$, one for each atom of the query that lead to a satisfied query, are extracted from all boolean variables $x^c_{q_e}$ that are reachable from boolean variable x_0 through a path of true boolean variables.

4 Application to JAVA Program Analysis

There is a strong interest in developing efficient demand-driven evaluation techniques that are applicable for program analysis since they naturally fit into *Integrated Development Environments* (IDEs) that dynamically provide analysis results to a programmer during the development of its code. Actually, demand-driven techniques are often considered better than global approaches during the program development since they usually encounter errors more rapidly by exploring only a portion of the code.

This also applies to Datalog queries: the more specific the query (*i.e.*, the higher the number of constant arguments), the better demand-driven resolution of the query, as compared to a global-based method, since only facts from the Datalog program that are necessary to answer the query will be inferred.

4.1 Datalog-based program analysis

The Datalog approach to static program analysis [18] can be summarized as follows. Each program element, namely variables, types, code locations, function names, are grouped in their respective *domains*. Thus, each argument $a_{i,j}$ of a predicate symbol p_i is typed by a domain $A_{i,j}$ of values. Hence, atoms $p_i : \wp(A_{i,j})^{n_i} \rightarrow \mathbb{B}$ are considered as *relations* among program's elements defined in their respective domains. By considering only finite program domains, Datalog programs are ensured to be *safe* (query evaluation generates a finite set of facts). Each program statement is decomposed into *basic program operations*, namely load, store, assignment, and variable declarations. Each kind of basic operation is described by a relation in a Datalog program. A program operation is then described as a tuple satisfying the corresponding relation.

Example 2. Consider the simple JAVA program [18] on the left-hand side of the following example:

```
public A foo { ...  p = new Object(); /* o1 */          vP_0(p, o1).
                    q = new Object(); /* o2 */    ⇒     vP_0(q, o2).
                    p.f = q;                           store(p, f, q).
                    r = p.f; ...  }                    load(p, f, r).
```

where `o1` and `o2` are heap allocations (extracted from corresponding bytecode). The Datalog pointer analysis approach consists first in extracting Datalog constraints (relations on the right-hand side of the above example) from the program. Then, it deduces further pointer-related information as output, like points-to relations vP from local variables and method parameters to heap objects as well as points-to relations hP between heap objects through field identifiers. The relation `vP_0` consists of initial points-to relations (v, h) of a program, *i.e.*, `vP_0(v,h)` holds if there exists a direct assignment within the program between a reference to a heap object `h` and a variable `v`. Other Datalog constraints such as `store` and `load` relations are calculated similarly.

In this framework, a *program analysis* consists in either querying extracted relations or computing new relations from existing ones. Datalog is both used to specify a static code analysis as well as to evaluate queries on given and inferred facts from the analysis.

Example 3. Consider the Datalog program that defines context-insensitive points-to analysis given in Fig. 1 (`pa.datalog` [18]).

```
### Domains
V 262144 variable.map
H 65536 heap.map
F 16384 field.map

### Relations
vP_0      (variable : V, heap : H)            inputtuples
store     (base : V, field : F, source : V)   inputtuples
load      (base : V, field : F, dest : V)     inputtuples
assign    (dest : V, source : V)              inputtuples
vP        (variable : V, heap : H)            outputtuples
hP        (base : H, field : F, target : H)   outputtuples

### Rules
vP (V1, H1)     :- vP_0(V1, H1).
vP (V1, H1)     :- assign(V1, V2), vP(V2, H2).
hP (H1, F1, H2) :- store(V1, F1, V2), vP(V1, H1), vP(V2, H2).
vP (V2, H2)     :- load (V1, F1, V2), vP(V1, H1), hP(H1, F1, H2).
```

Fig. 1. Datalog specification of a context-insensitive points-to analysis

The program consists of three parts:

1. A declaration of *domains*, with domain names and size (number of elements).
2. A list of *relations* specified by a predicate symbol, its arguments over specific domains and whether it is derived from an applicable Datalog rule (value `outputtuples`), or extracted from the program source code (value `inputtuples`).

3. A finite set of Datalog *rules*, defining the `outputtuples` relations.

A Datalog query consists of a set of goals over the relations defined in the Datalog program, *e.g.*, `:- vP(X,Y).` where `X` and `Y` are variable arguments of `vP`, meaning computing the complete set of variables in the domain of `X` that may point to any heap object `Y` at any point during program execution. From the initial relations of Example 2, we deduce by inferring the Datalog rules given by Fig. 1 the following output relations: `vP(p, o1), vP(q, o2), hP(o1, f, 02), and vP(r, o2)`. For instance, the output relation $vP(r, o2)$ indicates that variable r points to same heap allocation $o2$ as variable q.

4.2 Datalog-based program analyzer

We implemented the Datalog query transformation to BES in a powerful, fully automated Datalog solver tool, called DATALOG_SOLVE, developed within the CADP verification toolbox [8]. Without loss of generality, in this section, we describe the DATALOG_SOLVE tool focusing on JAVA program analysis. Other source languages and classes of problems can be specified in Datalog and solved by our tool.

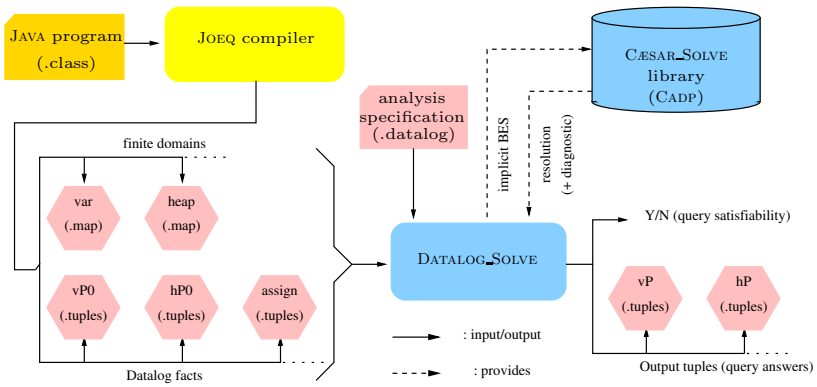

Fig. 2. JAVA program analysis using DATALOG_SOLVE tool

DATALOG_SOLVE takes three different inputs (see Fig. 2): the domain definitions (`.map`), the Datalog constraints or *facts* (`.tuples`), and a Datalog query $q = \langle G, R \rangle$ (`.datalog`, like `pa.datalog` in Fig. 1). The domain definitions state the possible values for each predicate's argument of the query. These are meaningful names for the numerical values that are used to efficiently described the Datalog constraints. For example, in the context of pointer analyses, variable names (`var.map`) and heap locations (`heap.map`) are two domains of interest. The Datalog constraints represent information relevant for the analysis. For instance, `vP0.tuples` gives all direct references from variables to heap objects in a given program. For efficiency reasons, these combinations are described by numerical values in the range $0..(domain\ size - 1)$.

Both, domain definitions and facts are specified in the `.datalog` input file (see Fig. 1) and they are automatically extracted from program source code by using the JOEQ compiler framework [17] that we slightly modified to generate *tuple-based* instead of BDD-*based* input relations.

DATALOG_SOLVE (120 lines of LEX, 380 lines of BISON and 3 500 lines of C code) proceeds in two steps:

1. The front-end of DATALOG_SOLVE constructs the optimized BES representation given by Equations 9-17 by extracting from the inputs (a particular analysis) the set of Datalog goals, rules and facts defining each boolean variable.
2. The back-end of our tool carries out the demand-driven generation, resolution and interpretation of the BES by means of the generic CÆSAR_SOLVE library of CADP, devised for local BES resolution and diagnostic generation.

This architecture clearly separates the implementation of Datalog-based static analyses from the resolution engine, which can be extended and optimized independently.

Upon termination (ensured by safe input Datalog programs), DATALOG_SOLVE returns both the query's satisfiability and the computed answers represented numerically in various output files (`.tuples` files). The tool takes as a default query the computation of the least set of facts that contains all the facts that can be inferred using the given rules. This represents the worst case of a demand-driven evaluation and computes all the information derivable from a Datalog program.

4.3 Experimental Results

The DATALOG_SOLVE tool was applied to a number of JAVA programs by computing the context-insensitive pointer analysis described above. To test the scalability

Table 1. Description of the JAVA projects used as benchmarks.

Name	Description	Classes	Methods	Vars	Allocs
freetts (1.2.1)	speech synthesis system	215	723	8K	3K
nfcchat (1.1.0)	scalable, distributed chat client	283	993	11K	3K
jetty (6.1.10)	server and servlet container	309	1160	12K	3K
joone (2.0.0)	Java neural net framework	375	1531	17K	4K

and applicability of the transformation, we applied our technique to four of the most popular 100% JAVA projects on Sourceforge that could compile directly as standalone applications and were used as benchmarks for the BDDBDDB tool [18]. They are all real applications with tens of thousands of users each. Projects vary in the number of classes, methods, variables, and heap allocations. The information details, shown on Table 1, are calculated on the basis of a context-insensitive callgraph precomputed by the JOEQ compiler. All experiments were conducted using JAVA JRE 1.5, JOEQ version

Table 2. Times (in seconds) and peak memory usages (in megabytes) for each benchmark and context-insensitive pointer analysis.

Name	time (sec.)	memory (Mb.)
freetts (1.2.1)	10	61
nfcchat (1.1.0)	8	59
jetty (6.1.10)	73	70
joone (2.0.0)	4	58

20030812, on a Intel Core 2 T5500 1.66GHz with 3 Gigabytes of RAM, running Linux Kubuntu 8.04.

The analysis time and memory usage of our context insensitive pointer analysis, shown on Table 2, illustrate the scalability of our BES resolution and validate our theoretical results on real examples. DATALOG_SOLVE solves the (default) query for all benchmarks in a few seconds. The computed results were verified by comparing them with the solutions computed by the BDDBDDB tool on the same benchmark of JAVA programs and analysis.

5 Conclusions and Future Work

This paper presents a novel approach to solve Datalog queries based on Boolean Equation System (BES) resolution and its application to program analysis. By using a local fixed point computation of BESs, our technique not only keeps the robustness of bottom-up over top-down evaluation semantics (problem of repeated computations and infinite loops), but also preserves the effectiveness of demand-driven techniques by taking advantage of constants and constraints that are part of the query's goals in order to reduce the search space. A new deductive database solver, called DATALOG_SOLVE, was designed and implemented, and several JAVA programs were analyzed modulo a context-insensitive pointer analysis encoded in Datalog. The tool architecture is based on the well-established verification framework CADP, which provides a generic library for local BES resolution.

As a future work, we plan to endow DATALOG_SOLVE with new, optimized strategies for local BES resolution, *e.g.*, rewriting of Datalog rules to allow goal-directed bottom-up evaluation, as in the *Magic sets* approach with complexity guarantees [14]. Another interesting improvement we plan to explore is to use the rewriting logic framework implemented in the reflective, functional programming language Maude [6] as a solver for Java program analyses using reflection. Finally, we could benefit from the regular structure of our BES encoding by distributing the BES resolution over a network of workstations with balanced partitioning preserving locality, similarly to the work of [9] that successfully applied a distributed BES resolution algorithm to numerous verification problems.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison Wesley, 1995.

2. H. R. Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1):3–30, 1994.
3. F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic Sets and Other Strange Ways to Implement Logic Programs. In *Proc. 5th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems PODS'86*, pages 1–15. ACM Press, 1986.
4. S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer, 1990.
5. T. Chen, B. Ploeger, J. van de Pol, and T. A. C. Willemse. Equivalence Checking for Infinite Systems Using Parameterized Boolean Equation Systems. In *Proc. 18th Int'l Conf. on Concurrency Theory CONCUR'07*, volume 4703 of *LNCS*, pages 120–135. Springer-Verlag, 2007.
6. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C.Talcott. *All About Maude: A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *LNCS*. Springer-Verlag, 2007.
7. A. van Dam, B. Ploeger, and T.A.C. Willemse. Instantiation for Parameterised Boolean Equation Systems. In *Proc. 5th Int'l Colloquium on Theoretical Aspects of Computing ICTAC'08*, LNCS. Springer-Verlag, 2008.
8. H. Garavel, R. Mateescu, F. Lang, and W. Serwe. CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes. In *Proc. 19th Int. Conf. on Computer Aided Verification CAV'07*, volume 4590 of *LNCS*, pages 158–163. Springer-Verlag, 2007.
9. C. Joubert and R. Mateescu. Distributed On-the-Fly Model Checking and Test Case Generation. In *Proc. 13th Int'l SPIN Workshop on Model Checking of Software SPIN'06*, volume 3925 of *LNCS*, pages 126–145. Springer-Verlag, 2006.
10. X. Liu and S. A. Smolka. Simple Linear-Time Algorithms for Minimal Fixed Points. In *Proc. 25th Int'l Colloquium on Automata, Languages, and Programming ICALP'98*, volume 1443 of *LNCS*, pages 53–66. Springer-Verlag, 1998.
11. R. Mateescu. Local Model-Checking of an Alternation-Free Value-Based Modal Mu-Calculus. In *Proc. 2nd Int'l Workshop on Verication, Model Checking and Abstract Interpretation VMCAI'98*, 1998.
12. R. Mateescu and D. Thivolle. A Model Checking Language for Concurrent Value-Passing Systems. In *Proc. 15th Int'l Symp. on Formal Methods FM'08*, volume 5014 of *LNCS*. Springer-Verlag, 2008.
13. T. W. Reps. Solving Demand Versions of Interprocedural Analysis Problems. In *Proc. 5th Int'l Conf. on Compiler Construction CC'94*, volume 786 of *LNCS*, pages 389–403. Springer-Verlag, 1994.
14. K. Tuncay Tekle, Katia Hristova, and Yanhong A. Liu. Generating specialized rules and programs for demand-driven analysis. In *Proc. 12th Int'l Conf. on Algebraic Methodology and Software Technology AMAST'08*, volume 5140 of *LNCS*. Springer-Verlag, 2008.
15. J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I and II, The New Technologies*. Computer Science Press, 1989.
16. L. Vieille. Recursive Axioms in Deductive Databases: The Query/Subquery Approach. In *Proc. 1st Int'l Conf. on Expert Database Systems EDS'86*, pages 253–267, 1986.
17. J. Whaley. Joeq: a Virtual Machine and Compiler Infrastructure. In *Proc. Workshop on Interpreters, Virtual Machines and Emulators IVME'03*, pages 58–66. ACM Press, 2003.
18. J. Whaley, D. Avots, M. Carbin, and M. S. Lam. Using Datalog with Binary Decision Diagrams for Program Analysis. In *Proc. Third Asian Symp. on Programming Languages and Systems APLAS'05*, volume 3780 of *LNCS*, pages 97–118. Springer-Verlag, 2005.
19. X. Zheng and R. Rugina. Demand-driven alias analysis for C. In *Proc. 35th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages POPL'08*, pages 197–208. ACM Press, 2008.